NAVAL POSTGRADUATE SCHOOL
Monterey, California

THESIS

DIMENSIONAL ANALYSIS OF
STRUCTURAL STEEL BEAM DESIGN

by

Michael A. Elizondo

March 1991

Thesis Advisor: Hemant K. Bhargava

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE

Form Approved OMB No. 0704-0188

1a REPORT SECURITY CLASSIFICATION: UNCLASSIFIED	1b RESTRICTIVE MARKINGS
2a SECURITY CLASSIFICATION AUTHORITY	3 DISTRIBUTION/AVAILABILITY OF REPORT: Approved for public release; distribution is unlimited
2b DECLASSIFICATION/DOWNGRADING SCHEDULE	
4 PERFORMING ORGANIZATION REPORT NUMBER(S)	5 MONITORING ORGANIZATION REPORT NUMBER(S)
6a NAME OF PERFORMING ORGANIZATION: Computer Technology Curr. Naval Postgraduate School	6b OFFICE SYMBOL (If applicable): 37
7a NAME OF MONITORING ORGANIZATION: Naval Postgraduate School	
6c ADDRESS (City, State, and ZIP Code): Monterey, CA 93943	7b ADDRESS (City, State, and ZIP Code): Monterey, CA 93943
8a NAME OF FUNDING/SPONSORING ORGANIZATION	8b OFFICE SYMBOL (If applicable)
9 PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER	
8c ADDRESS (City, State, and ZIP Code)	10 SOURCE OF FUNDING NUMBERS: PROGRAM ELEMENT NO; PROJECT NO; TASK NO; WORK UNIT ACCESSION NO

11 TITLE (Include Security Classification)
DIMENSIONAL ANALYSIS OF STRUCTURAL BEAM DESIGN (U)

12 PERSONAL AUTHOR(S)
Elizondo, Michael A.

13a TYPE OF REPORT	13b TIME COVERED	14 DATE OF REPORT (Year, Month Day)	15 PAGE COUNT
Master's Thesis	FROM _____ TO _____	March 1991	58

16 SUPPLEMENTARY NOTATION
The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or U.S. Government.

17 COSATI CODES			18 SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
FIELD	GROUP	SUB-GROUP	Dimensional Analysis, Modeling Languages

19 ABSTRACT (Continue on reverse if necessary and identify by block number)

This thesis examines the representation of dimensional units as prime numbers to perform dimensional analysis within a computer-based model management system. A computer program applies this concept to simple span structural steel beam design, an engineering stress and strain problem. Most common applications of computers manipulate only the numeric value of the measure of physical objects. The user manually ensures that data is processed according to the meaning of its units. Prime-encoding of dimensional units in this application provides a numeric method of validating dimensional consistency in mathematical expressions for use on a computer. This study is implemented in TEFA, a computer-based modeling system with an embedded Prolog programming language. The beam design application demonstrates that model representation using prime-encoding of dimensional units simplifies the overhead required in data manipulation, and helps maintain meaningful results in the numerical processing of data.

20 DISTRIBUTION/AVAILABILITY OF ABSTRACT	21 ABSTRACT SECURITY CLASSIFICATION
☒ UNCLASSIFIED/UNLIMITED ☐ SAME AS RPT ☐ DTIC USERS	UNCLASSIFIED
22a NAME OF RESPONSIBLE INDIVIDUAL: Hemant K. Bhargava	22b TELEPHONE (Include Area Code): (408) 646-2264 / 22c OFFICE SYMBOL: AS/Bh

DD Form 1473, JUN 86 — Previous editions are obsolete — S/N 0102-LF-014-6603

Approved for public release: distribution is unlimited.

Dimensional Analysis
of
Structural Steel Beam Design

by

Michael A. Elizondo
Lieutenant, United States Navy
B. Architecture, University of Texas at Austin, 1978

Submitted in partial fulfillment
of the requirements for the degree of

MASTER OF SCIENCE IN INFORMATION SYSTEMS

from the

NAVAL POSTGRADUATE SCHOOL

March 1991

Author: Michael A. Elizondo

Michael A. Elizondo

Approved by: H. K. Bhargava

Hemant K. Bhargava, Thesis Advisor

Gordon H. Bradley, Second Reader

David R. Whipple, Chairman
Department of Administrative Sciences

Accession For	
NTIS GRA&I	☑
DTIC TAB	☐
Unannounced	☐
Justification	
By	
Distribution/	
Availability Codes	
Dist	Avail and/or Special
A-1	

ABSTRACT

This thesis examines the representation of dimensional units as prime numbers to perform dimensional analysis within a computer-based model management system. A computer program applies this concept to simple span structural steel beam design, an engineering stress and strain problem. Most common applications of computers manipulate only the numeric value of the measure of physical objects. The user manually ensures that data is processed according to the meaning of its units. Prime-encoding of dimensional units in this application provides a numeric method of validating dimensional consistency in mathematical expressions for use on a computer. This study is implemented in TEFA, a computer-based modeling system with an embedded Prolog programming language. The beam design application demonstrates that model representation using prime-encoding of dimensional units simplifies the overhead required in data manipulation, and helps maintain meaningful results in the numerical processing of data.

TABLE OF CONTENTS

I. INTRODUCTION

A. OBJECTIVES

This thesis examines the representation of dimensional units as prime numbers to perform dimensional analysis within a computer-based model management system. A computer program applies this concept to simple span structural steel beam design, an engineering stress and strain problem. Most common applications of computers manipulate only the numeric value of the measure of physical objects [1: p. 478]. The user manually ensures that data is processed according to the meaning of its units. Prime-encoding of dimensional units in this application provides a numeric method of validating dimensional consistency in mathematical expressions for use on a computer [2: pp. 2-3]. This approach has been implemented in TEFA, a computer-based modeling system with an embedded Prolog programming language [3]. The beam design application demonstrates that:

> *Model representation using prime-encoding of dimensional units simplifies the overhead required in data manipulation, and helps maintain meaningful results in the numerical processing of data.*

Bradley [4: pp. 403-404] suggests that little attention has been given to representing and verifying data types used by computer programs. In most modeling applications, a computer-based mathematical model of a physical process typically manipulates the numerical portion of the model. However, the program may not be capable of distinguishing qualitative differences between the data it manipulates, making the program prone to errors. The goal of the modeler is to process numerical data according to its meaning, including constraints placed upon the data by the model. Bradley proposes that a system consisting of descriptions of model variables and rules for manipulating them can address this problem.

A simplified problem of structural steel beam design is used to demonstrate the representation of dimensional units as prime numbers in dimensional analysis. Beam design, in this thesis, is represented as a series of mathematical models describing the physical phenomena of a load upon a beam. In this context, a mathematical model is specified by inputs, objective functions, and constraints [5: pp. 2-7].

B. BENDING OF DUCTILE STEEL BEAMS

Parker [6: p. v] proposed that there is a continuing need to simplify structural steel design procedures for some types of buildings. Architects and engineers, henceforth referred to as *users*, require knowledge of engineering equations used for structural steel beam design to effectively coordinate design and construction. This includes validation of data available to the user for use in the appropriate equation. The beam design example is used to examine dimensional validation typically performed in an engineering problem.

Beam design begins with an examination of spanning and loading conditions of a building structure. Parker [6: p. 45-48] states that loads supported by beams are classified as either *concentrated* or *distributed* loads. A concentrated load is assumed to act at a point since it extends over a relatively small a portion of the beam length as shown in Figure 1.

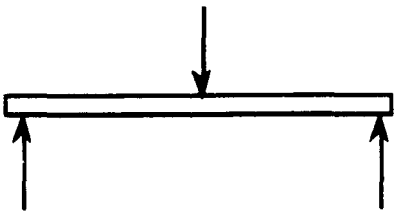


Figure 1 Simple Beam with Concentrated Load

A distributed load on a beam extends over a substantial portion of the entire length of a beam. Distributed loads are commonly referred to as *uniformly distributed*. The load has a uniform magnitude for each unit of length, such as pounds per linear foot or kips (1000 pounds) per linear foot, as shown in Figure 2.

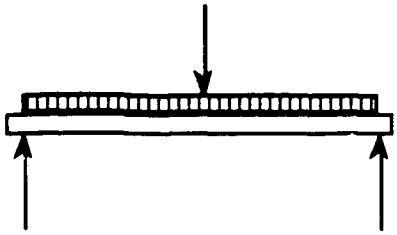


Figure 2 Simple Beam with Uniformly Distributed Load

The focus of this application is the design of a simple beam with a uniformly distributed load. A general procedure for this case of beam design is briefly outlined as follows [6: pp. 130-132]:

Step 1: Compute the load, W , in kips, that the beam will be required to support. If w is the uniformly distributed load in kip/feet, and L is the length in feet, then

$$W = w L.$$

Step 2: Compute the reaction, R, in kips:

$$R = W / 2.$$

Step 3: Compute the maximum bending moment, M, in kip-feet:

$$M = W L / 8.$$

Step 4: Determine section modulus in inches^3 by use of the beam formula where F_b = 24 ksi, the allowable bending stress of A36 steel:

$$S = M / F_b.$$

Step 5: Refer to the table of structural shapes, Properties for Designing, and select a beam with a section modulus S equal to or greater than that required by Step 4. See Appendix A. The compact section criteria in the table should confirm that $b_f / 2t_w$ is equal to or less than 8.70, the limiting value for A36 steel where F_b=24 ksi. The beam selected will be adequate for bending stresses subject to checking of section classification (compact or noncompact) and L_c or L_u controls.

Step 6: Check the beam for shear by confirming that the computed shearing stress $f_v = V / A_w$ in ksi is less than the allowable shear stress $F_v = 0.40 \times 36 \approx 14.5$ ksi for A36 steel. The static vertical shear on the beam, V in kips, is equal to R. The area of the girder web is determined by $A_w = d\ t_w$, in square inches, where d is the depth of the beam in inches and t_w is the web thickness of the beam in inches.

Step 7: Verify that the maximum actual deflection caused by the loading is less than the allowable deflection. Actual deflection is defined as follows:

$$\Delta = (5 \times f \times l^2) / (48 \times E \times c).$$

This simplifies to:

$$\Delta = .02483\, L^2)/\, d.$$

where Δ is in inches, f is in ksi, l = L x (12 in/ft) is in inches, E is the modulus of elasticity of steel equal to 29,000 ksi, and c is the distance from the neutral axis to the extreme fiber, where c = d/2 in inches. Allowable deflection Δ, in inches, is equal to 1/360 of the span or (L x 12)/360. If the actual Δ exceeds the allowable Δ, deflection may be solved for I, moment of inertia where I is in inches^4, required to limit Δ to the allowable value. This part of the procedure will not be attempted in this study.

Appendix B provides an example of a beam design problem using the above procedure. Appendix C provides a reference of general nomenclature used by the problem, and Appendix D provides a list of the abbreviations referenced in the design procedure. The purpose of the beam design application is to verify the dimensional consistency of the required mathematical expressions. The application also solves the expressions, recommending a wide flanged steel beam with a physical configuration fulfilling the design criteria of the problem.

C. DIMENSIONAL ANALYSIS OF BEAM DESIGN

Beam design illustrates typical difficulties of analyzing engineering expressions. Unit conversions are often required, as in the case of determining section modulus. The computed value for S, using the beam formula $S = M/F_b$, must be in inches^3 for comparison with section modulus values of the beam table. If the value of M is entered with dimensional units of kip-feet then M must be converted from kip-feet to kip-inches to usefully represent S in inches^3. The dimensional units of F_b can be represented as either ksi, kip/inches^2, or 1,000 pounds/inches^2, depending upon how data is presented to the user, or upon the user's preference. With these problems in mind, a fast, accurate, and automated system which performs checks of dimensional consistency is desirable.

D. OUTLINE OF THIS PAPER

In Chapter II, the concepts of measurement, physical algebra, dimensional analysis and application to modeling languages are briefly be discussed. Chapter III examines laws of dimensional consistency necessary to perform dimensional analysis of mathematical expressions. Prime-encoding of dimensional units is introduced for the purpose of performing dimensional analysis. Chapter IV presents the application of dimensional analysis and prime-encoding of dimensional units to the steel beam

design problem within a modeling language. Chapter V summarizes the results of the study and presents recommendations for further study. Appendices A through D provide a sample beam design problem with associated beam tables, nomenclature and abbreviations. Appendices E and F contain prime-encoded dimensional units and rules for determining dimensional consistency. Appendices G through J contain the beam design application, support processes and sample output. Appendix K describes the computer platform used to develop the application.

II. BACKGROUND

A. MEASUREMENT AND PHYSICAL ALGEBRA

Massey [7: p. 11-12] states that measurement is basically a comparison of things of the same kind. Measurement uses a unit defined as a standard amount of a quantity to compare with another of the same kind of quantity. The comparison is a magnitude with a numeric and a unit component. For example, to express the length of an object, a number quantifies the number of standard units equal to the length of the object, and a unit identifies the standard amount of that quantity.

The algebra used to describe describe physical objects, as suggested by Massey [7: pp. 12-13], is different from the algebra of pure mathematics. The algebra of pure mathematics, or ordinary algebra, is the expression of relations among numbers, though symbols are often used to represent numbers. Physical algebra describes relations among the magnitudes of physical quantities such as force, velocity, mass, energy, and so on. Physical algebra describes *how* the magnitude of one quantity depends on the magnitude of the others. Mathematical operations of addition, subtraction and comparison are restricted to quantities of the same kind. For example, a mass and an interval of time cannot be compared, but one mass and another mass can.

Massey [7: p. 13] states that physical algebra expresses relations between magnitudes of the same kinds of physical quantities. Magnitudes of the same kind of physical quantities are also expressed in terms of the same dimensional unit. When both sides of an equation have the same units, the equation is dimensionally homogeneous.

Relations in physical algebra, Massey [7: p. 13] asserts, must have two kinds of consistency. First, there is the quantitative relation of ordinary algebra in which both sides of an equation are compared in numerical magnitude. Second, there is a qualitative relation such that terms that are added, subtracted, or compared represent the same kind of quantity. Ordinary algebra is basically a means of comparing numbers, whereas physical algebra is a means of comparing the magnitudes of similar quantities.

B. DIMENSIONAL ANALYSIS

Langhaar [8: p. 1-4] describes dimensional analysis as a method of deducing information about physical phenomena if it can be described by a dimensionally correct equation. Scientific reasoning of the physical world is based on concepts of various abstract entities, such as force, mass, length, time, and so on. Each of these entities, or dimensions, may be assigned a unit measurement and is considered to be independent of the others.

Bhaskar [9: p. 73-74] states that dimensional analysis has been used in engineering for purposes of modeling and similitude. With dimensional analysis, reasoning about a system is possible without explicit knowledge of the physical laws that govern it. The model of the system requires knowledge of only relevant physical variables and their dimensional representation. Dimensional representation of physical variables contains a significant amount of knowledge.

Physical representations of variables, as stated by Bhaskar [9: p. 73-74] have both numerical and symbolic components. The numerical component is the value of a variable measured in a system of units. Reasoning about the numerical component represented by a physical variable is constrained by the physical context the value may take. The symbolic component in qualitative physics is the dimensional representation of the physical variables. For example, in the dimensional notation of physics, force is usually represented as pounds x feet x seconds^{-2}. Dimensional representation of a variable is also constrained by a set of laws. Dimensional homogeneity is the most familiar of these. One of the most widely used results of this concept in dimensional analysis is Buckingham's Π-Theorem, proved by Buckingham [10: pp. 345-376] in 1914. This theorem is used to establish the number of independent dimensionless numbers required to describe a given physical context [11: pp. 3918-3919]. Langhaar [8: p. 18] explains that according to Buckingham's Π-Theorem, an equation that relates dimensionless products is dimensionally homogeneous. Stated another way, the form of the equation is independent of the fundamental units of measurement.

C. DIMENSIONAL ANALYSIS USING A COMPUTER

Various computer-algebra systems and executable modeling languages [1: p. 478], [3], [4], [11], [12], [13] have been proposed to include symbolic physical units in computer

calculations. These techniqu offer automatic ion of dimensionally inhomogeneous formulas and tomatic conversion of nconsistent units in a dimensionally homogeneous formula.

Stoutemyer [1: p. 478] states that attempts to include physical units in mathematical expressions are sometimes abandoned since conventional programming languages generally deal with pure numbers rather than physical quantities. The user of the program must manually check data for its dimensional type to ensure that the results of the computation are valid. Errors of dimensional consistency are difficult to prevent and detect in traditional programming languages.

Stoutemyer [1: p. 478] suggested that since many programming languages have the ability to declare the precision of numerical variables and whether it is fixed or floating point, it should be possible to extend this technique to the declaration of variables such as units; i.e., meter, mile/hour, dollars, etc. A modeling language translator then checks expressions and assignments to variables for dimensional consistency relieving the user of the tedious, error-prone unit conversion process.

D. ALTERNATE METHODS

There are several computer programs incorporating dimensional analysis. Though they share the similar goal of validating dimensional consistency of a mathematical expression, they are quite different in their approach. The survey of alternative methods presented below is not exhaustive, but illustrates three different techniques of representing an extended data type containing numerical and dimensional information.

The first method, as described by Barnes [14: p. 3-14], is a database representation using a semantic type checking system for use with relational databases. The unit component of a variable, or semantic information, is associated with its numerical component in a data dictionary. This extended numeric data type, defined by concept, quantity and dimension, consists of a value description and a semantic description. The value description of the data type consists of a number and its unit of measure. The semantic description consists of a quantity and a concept, which is an object attribute possessing that quantity. Database queries are verified for dimensional consistency and dimensional units are converted across systems of measurement as necessary. Concepts are used to construct a concept hierarchy ensuring that queries are consistent

with the semantics of the database. This abstract numeric data type is used with the relational data model for dimensional validation and unit conversion.

Another variant of a computer-based dimensional analysis system proposed by Hirschberg [15: p. 2-9] uses the Buckingham Pi-Theorem [10] to order sets of equations by their importance. This system is implemented in SYMBOLANG, a list structured symbol manipulator based in the FORTRAN programming language. The symbol manipulator operates on a set of symbols rather than numbers. For example, a symbol manipulator multiplies N+1 by N-1 to obtain N^2-1. A list structure stores and manipulates data by defined relationships. Solution of the Pi-Theorem involves the formation of Pi-Terms ordering the equations by their importance. Once a Pi-Term is formed, the dimensionless set of terms is solved as a linear set of equations. The numeric solutions are then paired with their associated parameters. Hirschberg proposed that an extension of this work would include checking Pi-Term solutions against a set of well known dimensionless numbers. Data for the parameters would serve as input producing numeric answers. Hirschberg considers the speed of the computer an important advantage because it generates many permutations of equations yielding a large number of solutions, which can be solved by regression analysis for a best fit solution.

Stoutemyer [1: pp. 479-480] proposed yet another method using a computer-based symbolic algebra system to process dimensional units. The computer algebra technique is implemented in MACSYMA, developed by the MATHLAB Group. The method extends a programming language's ability to declare not only numerical variables, but also variables with a dimensional unit component such as meters, miles/hour, dollars, etc. This technique detects dimensionally inconsistent formulas and converts inconsistent units within a dimensionally homogeneous formula. A translator checks expressions and variables assignments for dimensional consistency.

E. DIMENSIONAL INFORMATION IN MODELING LANGUAGES

Having identified the need for verifying dimensional consistency of an equation describing physical relationships, it would be useful to have a flexible, easy to develop method of representing mathematical expressions describing the physical situations. Fourer suggests [13: pp. 143-155] that algorithmic solutions to solve problems within computer programs are explicit rather than symbolic. They are designed for

convenience and efficiency of handling data in a computer program rather than for clarity to the modeler. As a result of this approach, verificatic nd modification of the modeler's intentions become a problem of debugging computer programs. In addition, the description of the problem is highly dependent upon the form of the algorithm chosen to represent it.

Fourer [13: pp. 155-163] has argued modeling languages offer several advantages over algorithmic descriptions of a problem. The modeling language is not a programming language, but a declarative language that describes the modeler's intentions in a form that can be interpreted by a computer. The ability to describe the problem in the form of its mathem itical model provides inc ndence of any particular algorithmic form. Verific n is reduced to the tas debugging the modeling language's representation o. .ne problem. Bradley [12: p. 27] states that present research focuses on the use of modeling languages as executable : mputer programs. In general, modeling languages can be used to implement type calculus for dimensional systems. A variable in the model is assigned a type that consists of its concepts, quantities, and units of measurement. This representation permits checking the composition of expressions, and defining a hierarchy of concepts with inheritance of properties.

III. DIMENSIONAL ANALYSIS

A. INTRODUCTION

Dimensional analysis, as asserted by Massey [7: p. 108] and Langhaar [8: p. 1], cannot by itself provide a complete solution to a problem, but can provide a means for simplifying complex problems. Laws of dimensional consistency provide a means to check the validity of mathematical expression. Dimensional arithmetic provides a way to manipulate dimensional units. Together with prime encoding of dimensional units terms, as described by Bhargava [2] it is possible to develop a numeric process for executing these techniques in a computer program.

It is shown by Bhargava [2] that dimensional manipulation can be viewed as numerical arithmetic by recognizing the nature of dimensional arithmetic. Prime encoding supports this by representing each fundamental dimensional unit as a prime number. By the unique factorization theorem from number theory, numeric arithmetic applied to the prime encoding system follows the laws of dimensional arithmetic.

B. DIMENSIONAL ARITHMETIC OF UNIT MEASURES

Dimensional arithmetic provides a technique for manipulating the dimensional component of information [2: p. 3]. Dimensional arithmetic operates on dimensions in a similar manner to the way arithmetic operates on numbers.

A variable in a mathematical model may be defined by the quantity it measures and its dimensional unit. For example, in the beam problem, the section modulus of a beam, S in inches^3, is equal to moment, M in pound-inches, divided by bending stress of steel, F_b in pound/inches^2:

$$S = M / F_b.$$

The quantity of the variable generally has a base unit of measurement and may sometimes contain other dimensional units [4: p. 4]. These units are related to each other by laws of conversion within a system of measurement, such as one foot is equal to 12 inches. Dimensionless quantities without units, such as constants or ratios, may be represented simply by the number 1.

The International Metric System, SI, illustrates a standard system of seven fundamental quantities and base dimensional units [16].

UNITS FOR THE SI SYSTEM OF MEASURES

Quantity Measured	Base Unit	Other Units
Length	Meter	Kilometer, Centimeter
Mass	Kilogram	Gram, Milligram
Time	Second	Hour, Minute
Ampere	Electric Current	
Temperature	Kelvin	Celsius, Fahrenheit
Luminous Intensity	Candela	
Amount of Substance	Mole	

Dimensional units [2: p. 5-6] may be classified as fundamental or derived units. A unit is considered to be a fundamental unit if it is not a product of other units. A derived unit is product of other units.

Validation of dimensional information, as stated by Bhargava [2: pp. 3-6], requires dimensional manipulation which supports dimensional simplification and verification of the dimensional equivalency of expressions. Dimensional simplification reduces a mathematical expression to its simplest form. For example, the expression for section modulus, (pound x inch)/(pound x inch^2), simplifies to inch^3. Verification of dimensional equivalency requires recognition that one expression is dimensionally equivalent to another. For example, (pound x inch)/(pound / inch^2) is dimensionally equivalent to pound x inch x (1/pound) x (1/ inch) x (1/inch).

In summary, the following laws of dimensional consistency can be used for dimensional validation of expressions [2: pp. 6-7]:

1. Two expressions may be added or subtracted only if they are dimensionally equivalent.
2. Two expressions may be compared for equality or inequality only if their dimensions are equivalent.
3. Two expressions may be multiplied regardless of their dimensions.
4. An expression can be inverted regardless of its dimension.
5. The exponent of an expression must be dimensionless. It may be a fraction only if each dimensional unit in the expression has a power that is a multiple of that fraction, or if the expression is dimensionless.
6. Transcendental functions can be applied only to dimensionless expressions.

Dimensional arithmetic [2: p. 7] is used to perform the manipulations described above. The dimension of the sum or difference of two expressions is the same dimension of either only if the expressions have equivalent dimensions. If the expressions have different dimensions, the result is not defined. The dimension of the product or quotient of two expressions is the product or quotient of the dimensions of the two expressions respectively. An expression without a dimension has a dimension equal to 1. This is the dimensional multiplication identity. The dimension of an expression's exponent is the exponent of its dimension.

C. PRIME-ENCODING OF DIMENSIONS

The ability to represent dimensional unit terms as prime numbers provides a simple way of describing mathematical expressions within a computer-based modeling system. Dimensional unit terms can be uniquely represented as prime numbers, and derived dimensional unit terms can be represented by a combination of prime numbers. [2: pp. 9-10] The prime encoding of dimensional units can be illustrated by arbitrarily assigning fundamental dimensional units in a one-to-one correspondence with prime numbers. Since the number 1 is used as an identity, the sequence of prime numbers for this example will begin with 2. The numbers 2 and 3 are assigned to dimensional units, pound and inch, respectively.

Consider the expression for section modulus from the beam design problem:

$$S = M / F_b$$

or in dimensional unit terms only,

$$\text{inches}^3 = (\text{pound-inches}) / (\text{pounds}/ \text{inches}^2)$$

where S is section modulus in inches^3, M is moment of the beam in pound-inches, and F_b is bending stress in pounds/inches^2. The dimensional terms of M/ F_b simplify to inches^3, which verifies dimensional consistency. By use of the unique factorization theorem, also known as the fundamental theorem of arithmetic, it can be shown that dimensional equality is equivalent to numeric equality when the unit terms are expressed as primes numbers. [2: p. 10-11] For example, by substituting prime numbers for the dimensional units of M and F_b, the expression for section modulus becomes:

$$2^3 = (2 \times 3) / (3/ 2^2), \text{ or } 8 = 8.$$

D. SUMMARY

The application of prime number encoding of dimensional units, as proposed by Bhargava [2: pp. 11-12], suggests a method for performing dimensional manipulation. Prime numbers are arbitrarily assigned to a set of fundamental dimensional unit terms. The prime number 1 is reserved as the identity for multiplication. Dimensional multiplication and division are treated as numeric integer multiplication and division. Dimensional addition and subtraction are simply checked for equivalency. This yields a relatively simple method of dimensional manipulation. In summary, dimensional consistency and validity of an expression may be checked by computing the dimensional value of an algebraic e: ression, and by c puting the numeric value of the dimensional expression.

IV. IMPLEMENTATION

A. MODEL DEVELOPMENT

The beam design problem incorporates dimensional analysis through use of a modeling language. Modeling languages support development, documentation, and use of mathematical models [3: p. 1], [13: p. 144]. The modeling language chosen for this problem, TEFA [3], can represent mathematical models consisting of definitional equations. A modeling language approach permits concepts to be divided into modules easily shared with other applications. For example, a measurement system consisting of prime-encoded fundamental dimensional units, and the rules for dimensional consistency may be easily incorporated into other models describing other kinds of physical phenomena. See Appendices E and F [17].

Model development in a modeling language is similar to development of other abstract mathematical models. The purpose and the objectives of the model are clearly defined. Relationships between variables are stated mathematically. The mathematical models describing beam design can be conceptualized using a framework of variables and the mathematical relationships between them.

B. BACKGROUND

A practical outcome of the prime encoding of dimensional units, as suggested by Bhargava [2: pp. 2-4], [17], is a series of computer-programmable statements for testing dimensional consistency of mathematical expressions. Dimensional consistency is verified by performing the numerical equivalent of dimensional manipulation described by the expression. If an expression is dimensionally valid, the model described by the expression is dimensionally consistent. If the model is dimensionally consistent the test outlined in Appendix F returns a logical value of *true*. Otherwise, if the model is dimensionally inconsistent, the test returns a *false* value.

Numerical operations necessary to perform a dimensional consistency test requires recognition of dimensional operators. It also requires association of the dimensional unit terms with prime-encoded equivalents. The prime-encoded terms are operated upon according to the rules of dimensional manipulation. To exploit the

prime number representation of dimensional units, a function is defined which accepts a dimensional expression and returns a numeric value. See Appendix G. This value is computed from the dimensional operation on the prime-encoded dimensional unit terms:

numeric function(dimensional expression) → numeric value.

Assuming a database of prime values for fundamental units, it is possible to relate the prime-encoding of dimensional units. To implement the concept of prime-encoding of units, a table is constructed with fundamental dimensional unit terms with arbitrarily assigned prime numbers. See Appendix E [17: pp. 120-126]. The prime number is assigned to the dimensional unit by defining an additional rule for the numeric function:

numeric function(unit, prime number) → unit code(system, prime number, unit).

The numeric function returns the prime number indexed by the dimensional unit term. For example, assume that unit code(english measurement system, 2, inch). By numeric function(inch, prime number) the prime number value of 2 is returned.

This numeric function also includes rules for dimensional manipulation of the prime encoded dimensional units. To illustrate a general case of how this function performs dimensional addition:

numeric function (expression$_1$, n_1),
numeric function(expression$_2$, n_2),
$n_1 = n_2$,
numeric value is n_1
→ numeric function(expression$_1$ + expression$_2$, numeric value).

The numeric function evaluates the expressions for the prime number equivalent of their dimensional units. Note that this dimensional addition rule for the numeric function obeys the dimensional consistency laws. In a similar fashion, the remaining rules for subtraction, multiplication and division of dimensional unit operations are developed. See Appendix G.

An additional tool is necessary to process dimensional expressions into a form that can be analyzed for consistency or validity. This tool is a list function which processes each term of an expression. For example, a list processor for adding dimensional units where "+" is the dimensional addition operator has a form:

sum list([element$_1$ | element$_2$], answer) →
list(element$_1$, n_1),
list(element$_2$, n_2),
answer is $n_1 + n_2$.

The term, element$_2$, can also represent a list. This operation continues recursively until all elements of the list have been processed.

C. EXAMPLE: BENDING OF DUCTILE STEEL BEAMS

A primary objective of beam design is to make a preliminary selection of a beam with the smallest section modulus larger than the section modulus computed by the beam formula:

$$S = M / F_b, \text{ such that } F_b = 24 \text{ kip/in}^2 \text{ for A36 steel.}$$

To illustrate the implementation of this problem into a modeling language, the beam formula is expressed as a definitional equation:

definitional expression(compute section modulus, $S \leftarrow (M * 12) / F_b$).

The variable M, computed moment of the beam, is assumed to be defined in dimensional units of kip-feet. The definitional expression includes the conversion of M into kip-inches.

Qualitative information about the variable is described within a local variable declaration within the mathematical model. This declaration provides the variable name, the model name in which the variable is used, the local variable name within the model, information on whether the variable is computed internally or externally to the model, and also the storage type of the dimensional units of the variable. For example:

local variable(computed section modulus, compute section modulus, S, inch^3).

This information is mapped to a declaration to be represented by the variable. This notion is referred to as quiddity, or the essential quality of the variable. In the example of section modulus, this information tells the modeler that moment is a derived dimensional unit of $inch^3$:

quiddity variable(computed section modulus, volume, $inch^3$).

Information defined in these expressions can be shared with other models in a knowledge base. The measurement system in Appendix E is referenced to establish prime encoded dimensional units. Statement of the laws for dimensional consistency in Appendix F is referenced to verify the dimensional consistency of the mathematical expression defining section modulus for a beam. This verifies that the units in the mathematical expression are consistent with the way they are stored in the model. See Appendices I and J [3].

D. SUMMARY

The development of a model which incorporates dimensional validation, as described by Bhargava [3: pp. 7-8], consists of defining the mathematical equation, its variables and dimensional units, encoding the dimensional units with a set of prime numbers, and finally evaluating the numerical equivalent of the expression by of the laws defining dimensional consistency. Each equation of the beam design process is expressed as a mathematical model consisting of mathematically defined equations and variables. The model is concisely described and documented, simplifying its use by other models. Dimensional consistency is checked by the prime-encoding of the definitional equation's dimensional units. This technique verifies that variables used by the definitional equation are dimensionally valid.

V. RECOMMENDATIONS AND CONCLUSION

A. CONSTRAINTS

An important constraint of testing for dimensional consistency, as outlined by Bhargava [2: p. 15], is that the application selected for modeling must be subject to the standard laws of dimensional consistency. The beam design problem requires only simple substitution of prime-encoded dimensional units for validation of the subject mathematical expressions. The dimensional unit of the expressions describe physical measurements. A set of rules for the concept of "steel beam", or prime-encoding of "steel beam" types was not considered necessary. Abstract concepts, such as those suggested by Bradley [12: pp. 279-280], were not required by this problem to obtain a solution, so they were not explored in this study.

In addition, Dym [18: p. 1] states that the use of a model also places another important constraint on the problem solving process. Models are simply a mathematical representation of objects and processes. A mathematical model should not be confused with the reality of the physical problem. Mathematics performed in the context of a model are an abstraction of the actual physical problem which the model describes.

A constraint of the application itself is that it describes only simple span steel beams with a uniformly distributed load. Parker [6: pp. 45-84] proposes eight general cases of beam loading including concentrated loading, non-uniformly distributed loading, and cantilevered beams. Each case requires that different moments, shear, and deflections be considered for analysis. Stress parameters were considered only for A36 steel. Though A36 steel is the most commonly used material in steel frame building construction, there are several other steels available, each with their own allowable stress values. Additionally, only a small subset of fifteen W-shapes was considered for testing of the model. There are over sixty W-shapes available, as well as S-shapes, M-shapes, channels, compound angles, and bar-joists which can serve the same general function of a beam.

This system is implemented in Prolog and runs on Apple Macintosh II series of computers. TEFA [3], the model management system, is implemented in Prolog. See

Appendix K for additional information on the hardware and software utilized in this study.

B. RECOMMENDATIONS

The steel beam design application could be extended to overcome the constraints of the initial model. Additional models may be incorporated into the application covering other beam span conditions. This system could also be applied to an application serving as an engineer's aid in equation checking. This technique also has uses in fields such as operations research and finance for analyzing and checking parameters of various optimization models.

It would also be worthwhile to examine th easibility of using dimensional analysis and prime-encoding of dimensional uni n the background of another application, transparent to the user. For example, a CAD (Computer Aided Design) system may contain an expert system integrated into the application, providing interactive equation checking and dimensional validation.

This concept of dimensional checking may also be applied to other languages, such as conventional programming languages. It would be useful to examine the program structures necessary to parallel prime-encoding techniques of the Prolog language and dimensional consistency laws. This method may also be adaptable to spreadsheets by use of look-up tables and conditional statements describing the laws of dimensional consistency.

C. CONCLUSION

Bhargava [2: p. 14] points out that dimensional manipulation can be performed symbolically by existing symbolic mathematics programs. Bhargava proposes that the prime-encoding method of dimensional manipulation performs as well as other symbolic methods in retaining information about manipulated symbols. Units of measurement can be reconstructed from the numeric value representing the unit. This method is easy to implement in almost any programming language, making dimensional analysis possible in a wide range of applications. Since this system is designed to validate dimensional consistency of expressions, its application extends beyond numerical processing and d ase searching. The problems of dimensional consistency and dimensional validity ndled by the system so that the user is free to use a particular model across systems. The validity of the expressions

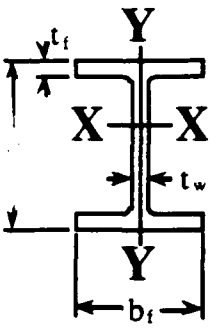
contained in the models are checked, and unit systems are converted as necessary to permit the passing of parameters between models. This ability addresses the long range problems of larger databases and maintenance of consistent data.

This thesis illustrates the feasibility of providing a computer-based numerical method for validating the dimensional consistency of a selected set of engineering equations. A fast, automatic method of validating the dimensional consistency of equations is demonstrated. Though one motivation of this system was to show the ability to convert units from one measuring system to another, it was not examined in this thesis. However, this system shows the usefulness of implementing a prime number encoding scheme for representation of units of measure in mathematical expressions using the inheritance properties of Prolog-type data structures. If governing laws for manipulation of dimensional formulas are not used, contradictions may result; for example, $in^3 = lb \times in^2$ [2], [19]. Such contradictions can be avoided by using standard conventions and techniques of analyzing dimensions recognizing the origin and derivation of formulas. Adherence to dimensional validation provides better understanding of models, and origins of mathematical expressions, and improvement of model representation.

APPENDIX A

PROPERTIES FOR DESIGNING

Selected Wide Flange Sections
W SHAPES
[6: p. 16]



Designation	Depth d (in.)	Flange Width b_f (in.)	Web thickness t_w (in.)	Elastic properties Axis X-X S ($in.^3$)	Compact section criteria $b_f/2t_f$
W10X89	10.88	10.275	0.615	99.7	5.15
W10X60	10.25	10.075	0.415	67.1	7.38
W10X49	10.00	10.000	0.340	54.6	8.96
W10X45	10.12	8.022	0.350	49.1	6.49
W10X39	9.94	7.990	0.318	42.2	7.75
W10X33	9.75	7.964	0.292	35.0	9.20
W10X25	10.08	5.762	0.252	26.5	6.70
W10X21	9.90	5.750	0.240	21.5	8.46
W8X67	9.00	8.287	0.575	60.4	4.44
W8X40	8.25	8.077	0.365	35.5	7.24
W8X31	8.00	8.000	0.288	27.4	9.24
W8X28	8.06	6.540	0.285	24.3	7.06
W8X24	7.93	6.500	0.245	20.8	8.17
W8X20	8.14	5.268	0.248	17.0	6.97
W8X17	8.00	5.250	0.230	14.1	8.52

APPENDIX B

EXAMPLE OF BEAM DESIGN PROCEDURE

Parker [6: pp. 127-132] describes a general design procedure for a simple span, laterally supported beam with a uniformly distributed load. The beam is designed with A36 steel. The allowable deflection is limited to 1/360 of the span. Loading of the beam is uniformly distributed.

Step 1: Compute the loads the beam will be required to support. For example, in a portion of a floor framing plan in Figure 1, a simple beam spans a distance of L = 10 feet and supports the floor a distance of 5 feet to either side of the beam. The floor load is 440 lb per sq ft, including the weight of the beam and other construction.

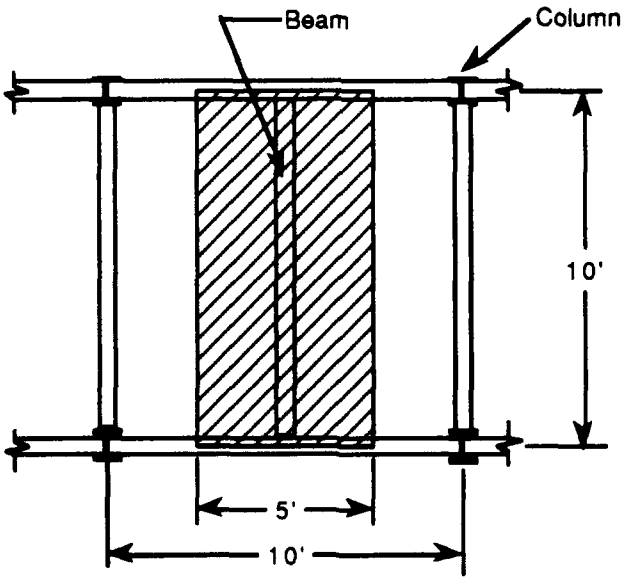


Figure 1 PORTION OF A FRAMING PLAN

The uniformly distributed load per linear foot, w, including the weight on the beam, is:

$$w = 5 \text{ ft} \times 440 \text{ lb per sq ft} = 2{,}200 \text{ lb/ft} = 2.2 \text{ kip/ft}.$$

The total uniformly distributed load, designated by W is the uniformly distributed load, w, times the span of the beam, L:

$$W = w \times L,$$

$$\text{or } W = (2.2 \text{ kips/ ft}) \times (10 \text{ ft}) = 22 \text{ kips}.$$

Step 2: Compute the reactions. Since the beam is symmetrically loaded as shown in Figure 2., $R_1 = R_2 = 22 \text{ kips}/2 = 11 \text{ kips}$.

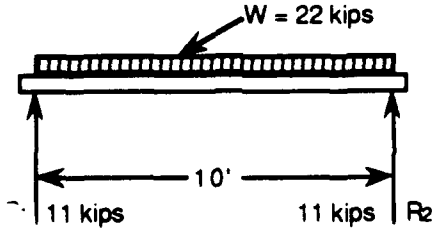


Figure 2 REACTION DIAGRAM

Step 3: Compute the maximum bending moment:

$$M = WL/8 = (22 \text{ kips} \times 10 \text{ ft})/8 = 27.5 \text{ kip-ft}.$$

Step 4: Determine the required section modulus by use of the beam formula $S = M/F_b$, where M is the bending moment in inch-pounds or kip-inches and F_b is the allowable bending stress in psi or ksi respectively. Since full lateral support is provided, the allowable bending stress of A36 steel is $F_b =$ 24 ksi if a compact section is used. (ksi = kips per square inch). Then

$$S = M/F_b = (27.5 \text{ kip-ft} \times 12 \text{ in per ft})/24 \text{ ksi} = 13.75 \text{ in}^3.$$

Step 5: Refer to the tables of structural shapes in Appendix A that give properties for designing, and select a beam with a section modulus equal to or greater than that required by Step 4. The beam selected will be adequate for bending stresses subject to checking of section classification (compact or noncompact) and L_c or L_u controls. Referring to Appendix A, [6: p. 16], we see that a W8X17 has a section modulus of 14.1 in^3, and we accept this as a trial beam. The compact section for the beam selection is checked. The compact section for A36 W-shapes is defined by AISC to be one in which the width-thickness ratio of the projecting compression flange, half-flange, does not exceed $52.2/\sqrt{F_y}$. The limiting value of the width-thickness ratio of the compression half-flange is computed to be $52.2/\sqrt{36} = 8.70$ for A36 steel [6: p. 22]. The compact section criteria in Appendix B shows that $b_f/2t_w = 8.52$ for the W8X17 section. Since this value is less than 8.7, the limiting value for A36 steel, the section is compact and the value of $F_b = 24$ ksi used in Step 4 is confirmed.

Step 6: Check the beam for shear by comparing the computed shearing stress $f_v = V/A_w$ with the allowable F_v. Checking the shear stress requires reference to Appendix A, which shows that d = 8 in and $t_w = 0.23$ in for the W8X17. Then since V = 11 kips:

$$f_v = V/A_w = 11 \text{ kips}/(8 \text{ in} \times 0.23 \text{ in}) = 5.97 \text{ ksi},$$

where A_w = area of girder web in sq in.

This value is less than the allowable 14.5 ksi, so the W8X17 is acceptable for shear.

Step 7: Compute the maximum deflection caused by the loading and compare it with the allowable delta, Δ. If the computed delta exceeds the allowable, the deflection formula may be solved for I required to limit delta to the allowable value [6: p. 83]. A beam section with an adequate moment of inertia may then be selected from the tables giving properties for designing. The allowable deflection is stated in the data as 1/360 of the span and is

(10 ft x 12 ft per in)/360 or 0.. . in. S e the allowable ding stress for this beam is 24 ksi, the following expression may be used to determine actual deflection [6: pp. 112-113, 131]:

$$\Delta = (5 \times f_b \times l^2) / (48 \times E \times c) = (0.02483\ L^2) / d.$$

$$\Delta = (0.02483 \times 10^2) / (8) = 0.31 \text{ in.}$$

The value is less than the allowable 0.33 in, so the W8X17 is acceptable for this loading. This expression is true only for simple steel beams with uniformly distributed loads. The extreme fiber stress, f_b, is 24,000 psi for A36 steel, and E = 29,000,000 psi.

APPENDIX C

NOMENCLATURE

A partial selection of nomenclature and terms [6: pp. 2-5]

AISC American Institute of Steel Construction, Inc.

A_w Area of girder web (sq in)

E Modulus of elasticity of steel (29,000 kips per sq in)

F_b Bending stress permitted in the absence of axial force (ksi or psi)

F_v Allowable shear stress (ksi or psi)

F_y Specified minimum yield stress of the type of steel being used (ksi or psi). As used in AISC (American Institute of Steel Construction) Specification, "yield stress" denotes either the specified minimum yield point (for those steels that have yield point) or specified minimum yield strength (for those steels that do not have a yield point).

I Moment of inertia of a section (in^4)

L Span length (ft)

L_c Maximum unbraced length of the compression flange at which the allowable bending stress may be taken at $0.66F_y$ (ft)

L_u Maximum unbraced length of the compression flange at which the allowable bending stress may be taken at $0.60\ F_y$ (ft)

M Moment (kip-ft or kip-in)

R Reaction (kips)

Maximum end reaction for 3-1/2 in. of bearing (kips)

S Elastic Section Modulus (in^3)

V Statical (vertical) shear on beams (kips)

W Total uniform load, including weight of beam (kips)

b_f Flange width of rolled beam or plate girder (in)

c Distance from the neutral axis to extreme fiber of beams (in)

d Depth of beam of plate girder (in)

f_b Computed bending stress (ksi or psi)

f_v Computed shear stress (ksi or psi)

l Actual unbraced length (in)

t_w Web thickness (in)

Δ Beam deflection (in)

w Uniformly distributed load per lineal foot (kip/lin ft)

APPENDIX D

ABBREVIATIONS

[6: p. 5]

Abbreviations	Quantity
cu ft	cubic foot
cu in	cubic inch
ft	foot
ft-lb	foot-pound
in	inch
in-lb	inch-pounds
kip	1000 pounds
kip-ft	kip-feet
kip-in	kip-inches
ksf	kips per square foot
ksi	kips per square inch
lin ft	linear foot
lb	pounds
lb per cu ft	pounds per cubic foot
lb per lin ft	pounds per linear foot
psf	pounds per square foot
psi	pounds per square inch
sq ft	square foot
sq in	square inch

APPENDIX E

MEASURES

[17: pp. 120-126]

```
/* measures.p created by hemant bhargava 1-20-89 */
/* Reference: Beyer's Mathematical Handbook */
/* measurement system, dimension, base unit, symbol, others */

/* Base units are fundamental or primar·    its. */
/* Units defined in terms of base units ar    ived units. */

/* base_unit(measurement_system, dimension, base_unit_symbol). */
  base_unit(si,length,mt).
  base_unit(si,mass,kg).
  base_unit(si,time,sec).
  base_unit(si,current,amp).
  base_unit(si,temperature,degK).
  base_unit(si,'luminous intensity',cd).
  base_unit(si,'amount of substance',mol).
  base_unit(si,currency,dollars).
  base_unit(si,volume,gal).

  base_unit(english,length,ft).
  base_unit(english,mass,lb).
  base_unit(english,time,sec).
  base_unit(english,current,amp).
  base_unit(english,temperature,degK).
  base_unit(english,'luminous intensity',cd).
  base_unit(english,'amount of substance',mol).
  base_unit(english,currency,pound).

/* unit( measurement_system, dimension, unit_symbol). */
  unit(si,length,mt).
  unit(si,length,cm).
  unit(si,length,km).
  unit(si,length,mm).

  unit(english,length,ft).
  unit(english,length,in).
  unit(english,length,yd).
  unit(english,length,mi).

  unit(si,area,mt^2).
  unit(si,volume,mt^3).
  unit(si,volume,lt).
  unit(si,volume,gal).
  unit(si,volume,qt).

  unit(si,mass,kg).
  unit(si,mass,gm).
```

```
unit(si,mass,mg).
unit(si,mass,tons).
unit(si,mass,ltons).

unit(english,mass,lb).
unit(english,mass,oz).
unit(english,mass,stons).

unit(_,time,sec).
unit(_,time,hr).
unit(_,time,min).
unit(_,time,day).

unit(_,temperature,degK).
unit(_,temperature,degC).
unit(_,temperature,degF).

unit(si,area,mt^2).
unit(si,volume,mt^3).
unit(si,volume,lt).
unit(english,area,ft^2).
unit(english,volume,ft^3).
unit(english,volume,floz).

unit_code(si,2,mt).
unit_code(si,3,cm).
unit_code(si,5,km).
unit_code(si,7,mm).

unit_code(english,11,ft).
unit_code(english,13,in).
unit_code(english,17,yd).
unit_code(english,19,mi).

unit_code(si,23,kg).
unit_code(si,29,gm).
unit_code(si,31,mg).
unit_code(si,37,tons).
unit_code(si,41,ltons).

unit_code(english,43,lb).
unit_code(english,47,oz).
unit_code(english,53,stons).

unit_code(_,61,sec).
unit_code(_,67,hr).
unit_code(_,71,min).
unit_code(_,73,day).

unit_code(_,83,degK).
unit_code(_,89,degC).
unit_code(_,91,degF).

unit_code(_,101,dollars).

/* The convention is to state across-system laws
      only for the base unit in each system
```

```
      e.g. between mt and ft, but not cm and ft.
      within system conversion is done using the conversion
      laws with the base unit as the meeting point */

/* Base conversion laws (ACROSS systems) */
base_conv_law(mt = 0.3048 * ft).
base_conv_law(kg = 0.45359237 * lb).

/*  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%  */
/* Conversion from base unit of a system to other unit in it */
/* The query we are interested in answering is
 C = ? F = ? in: conv_law(U1 = C + F * U2).  */
/* length */
conv_law(mt = 100 * cm).
conv_law(mt = 0.001 * km).
/* conv_law(mt = 1609.344 * mi). */
conv_law(mt = 1000 * mm).

/* length */
conv_law(ft = (1/3) * yd).
conv_law(ft = (1/5280) * mi).
conv_law(ft = 12 * in).

/* mass */
conv_law(kg = 1000 * gm).
conv_law(kg = 1000000 * mg).
conv_law(kg = (1/1000) * tons).
conv_law(kg = (1/1000) * ltons).

/* mass */
conv_law(lb = 16 * oz).
conv_law(lb = (1/2000) * stons).

/* time */
conv_law(sec = (1/60) * min).
conv_law(sec = (1/3600) * hr).
conv_law(sec = (1/86400) * day).

/* temperature */
conv_law(degK = 273.15 + degC).
conv_law(degF = 32 + (9/5) * degC).
conv_law(degC = (-32 * 5 / 9) + (5/9) * degF).

conv_law(mt^3 = 1000 * lt).
conv_law(gal = 4 * qt).

conv_law(U = 0 + 1 * U).
conv_law(U1^A = 0 + Factor * (U2^A)) :-
      conv_law(U1 = Factor * U2),
        Factor is pow(F,A).

/*  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%  */

/* For converting betwee  ase unit */
conv_law(BU1 = 0 + F12 *  J2) :-
      base_conv_law(BU1 = F12 * BU2).
```

```
conv_law(BU1 = 0 + F12 * BU2) :-
  base_conv_law(BU2 = F21 * BU1),
      F12 is 1 / F21.

conv_law(BU1 = F12 + 1 * BU2) :-
      base_conv_law(BU1 = F12 + BU2).
conv_law(BU1 = F12 + 1 * BU2) :-
  base_conv_law(BU2 = F21 + BU1),
      F12 is (-1) * F21.

/* Within system: base unit to other unit*/
conv_law(BU = 0 + F * U) :-
      conv_law(BU = F * U).
conv_law(BU = C + 1 * U) :-
      conv_law(BU = C + U).

/* For arbitrary conversions (within or across systems) */
conv_law(U1 = 0 + F12 * U2) :-
      unit(S1,Dimension,U1),
      base_unit(S1,Dimension,BU1),
      conv_law(BU1 = 0 + F1 * U1),
      !,
      unit(S2,Dimension,U2),
      base_unit(S2,Dimension,BU2),
      conv_law(BU2 = 0 + F2 * U2),
      conv_law(BU1 = 0 + BF12 * BU2),
      F12 is F2 * BF12 / F1.

/* For arbitrary conversions (within or across systems) */
conv_law(U1 = F12 + 1 * U2) :-
      unit(S1,Dimension,U1),
      base_unit(S1,Dimension,BU1),
      conv_law(BU1 = F1 + 1 * U1),
      !,
      unit(S2,Dimension,U2),
      base_unit(S2,Dimension,_,BU2),
      conv_law(BU2 = F2 + 1 * U2),
      conv_law(BU1 = BF12 + 1 * BU2),
      F12 is F2 + BF12 - F1.

unit_alias(meters,mt).
unit_alias(kilograms,kg).
unit_alias(seconds,sec).
unit_alias(amperes,amp).
unit_alias('degree Kelvin',degK).
unit_alias(candela,cd).
unit_alias(mole,mol).
unit_alias(dollars,dollars).

/* convert from units 2 to units 1 */
 convert(Number,Units1,Number,Units1) :- !.
 convert(Number2,Units2,Number1,Units1) :-
      conv_law(Units2 = F1 + F2 * Units1),
      Number1 is F1 + (F2 * Number2).
```

APPENDIX F

LAWS FOR DIMENSIONAL CONSISTENCY

[17]

```
/* Laws for dimensional consistency of functional and conditional expressions. */

/* dim_valid(M, Y =: Phi <-- Psi) :- units(M, Y, U1),
        units(M, Phi, U2),
        units_eq(U1, U2, true). */

dim_valid(M,Y =: Phi) :- units(M,Y,U1), units(M,Phi,U2  nits_eq(U1,U2,true).

dim_valid(M, A+B) :- units(M, A, DA),
        units(M, B, DB),
        units_eq(DA, DB, true).

dim_valid(M, A-B) :- units(M, A, DA),
        units(M, B, DB),
        units_eq(DA, DB, true).

dim_valid(M, A*B) :- dim_valid(M, A),
        dim_valid(M, B).

dim_valid(M, A/B) :- dim_valid(M, A),
        dim_valid(M, B).

dim_valid(M, A^B) :- dim_valid(M, A),
        dim_valid(M, B).

dim_valid(M, pow(A, B))    lim_valid  . A),
        dim_valid(M, B).

dim_valid(M,sum(_, X)) :- dim_valid(M, X).

dim_valid(M,prod(_, X)) :- dim_valid(M, X).

dim_valid(M, A=B) :- units(M, A, DA),
        units(M, B, DB),
        units_eq(DA, DB, true).

dim_valid(M, A<B) :- units(M, A, DA),
        units(M, B, DB),
        units_eq(DA, DB, true).

dim_valid(M, A>B) :- units(M, A, DA),
        units(M, B, DB),
        units_eq(DA, DB, true).

dim_valid(M, A=<B) :- units(  A, DA),
        units(M, B, DB),
```

```
        units_eq(DA, DB, true).

dim_valid(M, A>=B) :- units(M, A, DA),
        units(M, B, DB),
        units_eq(DA, DB, true).

dim_valid(M, Exp, false).

/* Laws for Computing derived units */


units(M, A+B, UA) :- units(M, A, UA),
        units(M, B, UB),
        units_eq(UA, UB, true).

units(M, A-B, UA) :- units(M, A, UA),
        units(M, B, UB),
        units_eq(UA, UB, true).

units(M, A*B, U) :- units(M, A, UA),
        units(M, B, UB),
        simplify(UA*UB, U).

units(M, A/B, U) :- units(M, A, UA),
        units(M, B, UB),
        simplify(UA/UB, U).

units(M, A^B, U) :- units(M, A, UA),
        units(M, B, UB),
        simplify(UA^UB, U).

units(M, pow(A, B), U) :- units(M, A, UA),
        units(M, B, UB),
        simplify(UA^UB, U).

units(M,max(_,X), U) :- units(M, X, U).

units(M,min(_,X), U) :- units(M, X, U).

units(M,sum(_,X), U) :- units(M, X, U).

/* units(M, prod(_,X), U) :- units(M, X, U). requires multiplication of each */

units(M, Local_var, U) :- local_var(_, M, Local_var, _, U).

units_eq(A, A, true).

/* If any of the definitional expressions is dimensionally invalid, the model is not d.c. */

/* If a model is d.c. the second argument of the dim_valid_model predicate will be 'true'.
   If it is not, the second argument will contain the invalid expression. */

dim_valid_model(Model, DefE) :- defE(Model, _, DefE),
    not(dim_valid(M, DefE)), !.

/* If any of the conditional expressions is dimensionally invalid, the model is not d.c. */
```

```
dim_valid_model(Model, ConE) :- defE(Model, _, ConE),
    not(dim_valid(M, ConE)), !.

/* Else it must be OK */

dim_valid_model(Model, true).
```

APPENDIX G

A NUMERIC METHOD

```
/* numeric.p */
/* 12 Feb 1991 */
/* Get numeric value for derived units. */
/* Assume database of prime values for fundamental units*/

/* numeric(Dimensional_expression, Numeric_value) */

        numeric(Exp_1*Exp_2, Numeric_value) :-
                numeric(Exp_1, N1),
                numeric(Exp_2, N2),
                Numeric_value is N1*N2.

        numeric(Exp_1/Exp_2, Numeric_value) :-
                numeric(Exp_1, N1),
                numeric(Exp_2, N2),
                Numeric_value is N1/N2.

        numeric(Exp_1+Exp_2, Numeric_value) :-
                numeric(Exp_1, N1),
                numeric(Exp_2, N2),
                N1 = N2,
                Numeric_value is N1.

        numeric(Exp_1-Exp_2, Numeric_value) :-
                numeric(Exp_1, N1),
                numeric(Exp_2, N2),
                N1 = N2,
                Numeric_value is N1.

        numeric( Exp_1^Exp_2, Numeric_value) :-
                numeric(Exp_1, N1),
                numeric(Exp_2, N2),
                N2 = 0,
                Numeric_value is N1.

        numeric(Unit, Prime) :- unit_code(System, Prime, Unit).

        numeric(X, 1) :- number(X).
```

APPENDIX H

A LIST PROCESS

```
/* multiply_list([[A]|[B]], Answer). */
multiply_list([], 1).

multiply_list([A|B], Answer) :-
        multiply_list(A, X),
        multiply_list(B, Y),
        Answer is X*Y.

multiply_     A, Ans ) :-
        nu...er(A),
        Ans is A.

/* divide_list([[A]|[B]], Answer). */
divide_list([], 1).

divide_list([A|B], Answer) :-
        divide_list(A, X),
        divide_list(B, Y),
        Answer is X/Y.

divide_list(A, Ans ) :-
        number(A),
        Ans is A.

/* add list */
/* add_list([[A]|[B, C, D]], Answer). */
/* add_list([A|[B, C, D]], Answer). */
add_list([], 0).

add_list([A|B], Answer) :-
        add_list(A, X),
        add_list(B, Y),
        Answer is X+Y.

add_list(A, Answer) :-
        number(A),
        Answer is A.

/* subtract_list([[A]|[B]], Answer). */
subtract_list([], 0).

subtract_list([A|B], Answer) :-
        subtract_list(A, X),
        subtract_list(B, Y),
        Answer is X-Y.
```

```
subtract_list(A, Ans ) :-
        number(A),
        Ans is A.
```

APPENDIX I

BEAM MODEL

```
/* model(model_name, description). */
/* model_source(model_name, model_reference). */
/* variable(variable_name, description). */
/* var_quiddity(variable_name, type_of_quantity, quiddity, storage_units). */
/* defE(model_name, expression_number, definitional expression). */
/* local_var(model_name, variable_name, local_name, <exo or endo>, units). */
/* setDef(model_name, index_name, 'description', index_range). */
/* indexvar(index, model_name, 'description', [type(index), index_range]). */
/* in_model(model_name, use(model_name, local_variable),
        to_eval(local_variable)). */
/* run_report(model_name, [variable_list]). */
/* scenario(model_name(scenario_number), description). */
/* m_s_pair(model_name, model_name(scenario_number)). */
/* datum(variable_name, model_name(scenario_number), data). */


/*Variable base */

var_quiddity(allowable_shear_stress, mass/area, *, kip/in^2).
        variable(allowable_shear_stress, 'Allowable shear stress of beam').

var_quiddity(actual_deflection, length, *, in).
        variable(actual_deflection, 'Actual deflection of beam').

var_quiddity(allowable_deflection, length, *, in).
        variable(allowable_deflection, 'Allowable deflection of beam').

var_quiddity(beam_depth#i, length, *, in
        var(beam_depth#i, ['Depth of beam', i]).

var_quiddity(bending_stress, mass/area, *, kip/in^2).
        variable(bending_stress, 'Bending stress').

var_quiddity(computed_section_modulus, volume, *, in^3).
        variable(computed_section_modulus, 'Computed section modulus from beam
formula S=M/Fb').

var_quiddity(computed_shear_stress, mass/area, *, kip/in^2).
        variable(computed_shear_stress, 'Computed shear stress').

var_quiddity(depth_solution, length, *, in).
        variable(depth_solution, 'Depth of wide flange solution').

var_quiddity(length_of_beam, length, *, ft).
        variable(length_of_beam, 'Length of beam').
```

```
var_quiddity(max_vertical_shear, mass, *, kip).
        variable(max_vertical_shear, 'Maximum vertical shear of beam').

var_quiddity(moment_of_beam, mass*length, *, kip*ft).
        variable(moment_of_beam, 'Moment of beam ).

var_quiddity(reaction, mass, *, kip).
        variable(reaction, 'Reaction of beam').

var_quiddity(section_modulus#i, volume, *, in^3).
        variable(section_modulus#i, ['Table section modulus ', i]).

var_quiddity(uniformly_distributed_load, mass/length, *, kip/ft).
        variable(uniformly_distributed_load, 'Uniformly distributed load').

var_quiddity(w_shape#i, _, *, _).
        variable(w_shape#i, ['Wide flange shape ', i]).

var_quiddity(web_thickness#i, length,*, in).
        variable(web_thickness#i, ['Web thickness of beam ', i]).

var_quiddity(web_thickness_solution, length,*, in).
        variable(web_thickness_solution, 'Web thickness of wide flange solution').

var_quiddity(weight_of_load, mass, *, kip).
        variable(weight_of_load, 'Weight of load').

var_quiddity(wide_flange_solution, _, *, _).
        variable(wide_flange_solution, 'Wide flange solution').



model(beam, 'beam: Model for selection of simple span beams.').
model_source(beam,'Parker: Simplified Design of Structural Steeel.').
m_s_pair(beam, beam(1)).



/* step_1: Compute weight */
model(step_1, 'Compute weight of floor load.').
model_source(step_1,'Parker').
m_s_pair(step_1, beam(1)).
        local_var(weight_of_load, step_1, cap_W, endo, kip).
        local_var(length_of_beam, step_1, cap_L, exo, ft).
        local_var(uniformly_distributed_load, step_1, w, exo, kip/ft).
/* weight_of_load(kip) = uniformly_distributed_load(kip/ft) * length of beam(ft) */
defE(step_1, 1, cap_W =: w * cap_L).
        in_model(step_1, use(step_1, cap_W), to_eval(cap_W)).
run_report(step_1, [cap_W, w, cap_L]).
```

```
/* step_2: Compute reactions */
model(step_2, 'Compute beam reactions.').
model_source(step_2,'Parker').
m_s_pair(step_2, beam(1)).
        local_var(reaction, step_2, cap_R, endo, kip).
        local_var(max_vertical_shear, step_2, cap_V, endo, kip).
        local_var(weight_of_load, step_2, cap_W, exo, kip).
/* reaction(kip) = (weight_of_load(kip) / 2) */
defE(step_2, 2-1, cap_R =: (cap_W / 2)).
/* max_vertical_shear(kip) = reaction(kip) */
defE(step_2, 2-2, cap_V =: cap_R).
        in_model(step_2, use(step_1, cap_W), to_eval(cap_W)).
        in_model(step_2, use(step_2, cap_R), to_eval(cap_R)).
        in_model(step_2, use(step_2, cap_V), to_eval(cap_V)).
run_report(step_2, [cap_R, cap_V]).



/* step_3: Compute moment */
model(step_3, 'Compute moment.').
model_source(step_3,'Parker').
m_s_pair(step_3, beam(1)).
        local_var(moment_of_beam, step_3, cap_M, endo, kip*ft).
        local_var(length_of_beam, step_3, cap_L, exo, ft).
        local_var(weight_of_load, step_3, cap_W, exo, kip).
/* moment_of_beam(kip*ft) = (weight_of_load(kip)) * length_of_beam(ft))/ 8 */
defE(step_3, 3, cap_M =: (cap_W * cap_L)/8).
        in_model(step_3, use(step_1, cap_W), to_eval(cap_W)).
        in_model(step_3, use(step_2, cap_R), to_eval(cap_R)).
        in_model(step_3, use(step_2, cap_V), to_eval(cap_V)).
        in_model(step_3, use(step_3, cap_M), to_eval(cap_M)).
run_report(step_3, [cap_M]).



/* step_4: Compute section modulus */
model(step_4, 'Compute section modulus').
model_source(step_4,'Parker').
m_s_pair(step_4, beam(1)).
        local_var(computed_section_modulus, step_4, cap_S, endo, in^3).
        local_var(moment_of_beam, step_4, cap_M, exo, kip*ft).
        local_var(bending_stress, step_4, cap_Fb, exo, kip/in^2).
/* computed_section_modulus(in^3) = (moment_of_beam(kip*ft) * (12
in/ft))/bending_stress(kip/in^2)) */
defE(step_4, 4, cap_S =: (cap_M *12) / cap_Fb).
        in_model(step_4, use(step_1, cap_W), to_eval(cap_W)).
        in_model(step_4, use(step_2, cap_R), to_eval(cap_R)).
        in_model(step_4, use(step_2, cap_V), to_eval(cap_V)).
        in_model(step_4, use(step_3, cap_M), to_eval(cap_M)).
        in_model(step_4, use(step_4, cap_S), to_eval(cap_S)).
run_report(step_4, [cap_S, cap_Fb]).
```

```
/* step_5: Select beam */
model(step_5, 'Select beam using computed section modulus').
model_source(step_5,'Parker').
m_s_pair(step_5, beam(1)).
%       local_var(solution_list, step_5, [list], endo, _).
        local_var(test_variable, step_5, test, endo, _).
        local_var(wide_flange_solution, step_5, w_sol, endo, _).
        local_var(depth_solution, step_5, d, endo, in).
        local_var(web_thickness_solution, step_5, tw, endo, in).
        local_var(computed_section_modulus, step_5, cap_S, exo, in^3).
        local_var(section_modulus, step_5, s_mod, exo, in^3).
        local_var(w_shape#i, step_5, shape#i, exo, _).
        local_var(beam_depth#i, step_5, depth#i, exo, in).
        local_var(web_thickness#i, step_5, thick#i, exo, in).
% defE(step_5, 5-1, list=: vector(i in a_list, i <-- s_mod#i >= cap_S)).
% defE(step_5, 5-2, w_sol =: min_list(list)).
defE(step_5, 5-3, w_sol =: shape#15).
defE(step_5, 5-4, d =: depth#15).
defE(step_5, 5-5, tw=: thick#15).
%       setDef(step_5, a_list, 'Possible beam choices', 1...15).
%       indexvar(i, step_5, 'Beam choice index', [i in a_list]).
        indexvar(i, step_5, 'Beam choice index', [integer(i), 1...15]).
        in_model(step_5, use(step_4, cap_S), to_eval(cap_S)).
        in_model(step_5, use(step_5, w_sol), to_eval(w_sol)).
        in_model(step_5, use(step_5, d), to_eval(d)).
        in_model(step_5, use(step_5, tw), to_eval(tw)).
%       in_model(step_5, use(step_5, list), to_eval(list)).
        in_model(step_5, use(step_5, test), to_eval(test)).
run_report(step_5, [w_sol, d, tw]).



/* Step_6: Check shear stress */
model(step_6, 'Check shear stress').
model_source(step_6,'Parker').
m_s_pair(step_6, beam(1)).
        local_var(computed_shear_stress, step_6, fv, endo, kip/in^2).
        local_var(shear_check, step_6, shear, endo, _).
        local_var(allowable_shear_stress, step_6, cap_Fv, exo, kip/in^2).
        local_var(max_vertical_shear, step_6, cap_V, exo, kip).
        local_var(depth_solution, step_6, d, exo, in).
        local_var(web_thickness_solution, step_6, tw, exo, in).
/*computed_shear_stress(kip/in^2)=max_vertical_shear(kip)/(depth(in) * thick(in)).*/
defE(step_6, 6-1, fv =: cap_V / (d * tw)).
/* If allowable _shear_stress(kip/in^2) >= computed_shear_stress(kip/in^2) then
(shear_check = ok) */
defE(step_6, 6-2, shear =: (ok <-- cap_Fv >= fv)).
        in_model(step_6, use(step_2, cap_V), to_eval(cap_V)).
        in_model(step_6, use(step_5, d), to_eval(d)).
        in_model(step_6, use(step_5, tw), to_eval(tw)).
        in_model(step_6, use(step_6, fv), to_eval(fv)).
        in_model(step_6, use(step_6, shear), to_eval(shear)).
run_report(step_6, [shear, fv, cap_Fv]).
```

```
/* Step_7: Check deflection */
model(step_7, 'Check deflection').
model_source(step_7,'Parker').
m_s_pair(step_7, beam(1)).
        local_var(allowable_deflection, step_7, allow, endo, in).
        local_var(actual_deflection, step_7, actual, endo, in).
        local_var(deflection_check, step_7, defl, endo, _).
        local_var(length_of_beam, step_7, cap_L, exo, ft).
        local_var(depth_solution, step_7, d, exo, in).
/* allowable_deflection(in) = (length_of_beam(ft) * (12 in/ft))/ 360) */
defE(step_7, 7-1, allow =: (cap_L * 12) / 360).
/* actual_deflection(in) = (0.02483(in^2/ft^2) * length_of_beam(ft^2) / d(in) */
defE(step_7, 7-2, actual =: (0.02483 * cap_L^2 ) / d).
/* If actual_deflection(in) =< allowable_deflection(in) then (deflection_check = ok) */
defE(step_7, 7-3, defl =: (ok <-- actual =< allow)).
        in_model(step_7, use(step_5, d), to_eval(d)).
        in_model(step_7, use(step_7, allow), to_eval(allow)).
        in_model(step_7, use(step_7, actual), to_eval(actual)).
        in_model(step_7, use(step_7, defl), to_eval(defl)).
run_report(step_7, [defl, actual, allow]).


scenario(beam(1), 'This is a test scenario of the beam model').


datum(allowable_shear_stress, beam(1), 14.5).
datum(bending_stress, beam(1), 24.0).
datum(length_of_beam, beam(1), 10.0).
datum(uniformly_distributed_load, beam(1), 2.2).

/* Table 1-1, Parker, Simplified Design of Structural Steel */

datum(w_shape#1, beam(1), w10_89).
datum(w_shape#2, beam(1), w10_60).
datum(w_shape#3, beam(1), w10_49).
datum(w_shape#4, beam(1), w10_45).
datum(w_shape#5, beam(1), w10_39).
datum(w_shape#6, beam(1), w10_33).
datum(w_shape#7, beam(1), w10_25).
datum(w_shape#8, beam(1), w10_21).
datum(w_shape#9, beam(1), w8_67).
datum(w_shape#10, beam(1), w8_40).
datum(w_shape#11, beam(1), w8_31).
datum(w_shape#12, beam(1), w8_28).
datum(w_shape#13, beam(1), w8_24).
datum(w_shape#14, beam(1), w8_20).
datum(w_shape#15, beam(1), w8_17).

datum(section_modulus#1, beam(1), 99.7).
datum(section_modulus#2, beam(1), 67.1).
datum(section_modulus#3, beam(1), 54.6).
datum(section_modulus#4, beam(1), 49.1).
datum(section_modulus#5, beam(1), 42.2).
datum(section_modulus#6, beam(1), 35.0).
datum(section_modulus#7, beam(1), 26.5).
```

```
datum(section_modulus#8, beam(1), 21.5).
datum(section_modulus#9, beam(1), 60.4).
datum(section_modulus#10, beam(1), 35.5).
datum(section_modulus#11, beam(1), 27.4).
datum(section_modulus#12, beam(1), 24.3).
datum(section_modulus#13, beam(1), 20.8).
datum(section_modulus#14, beam(1), 17.0).
datum(section_modulus#15, beam(1), 14.1).

datum(beam_depth#1, beam(1), 10.88).
datum(beam_depth#2, beam(1), 10.25).
datum(beam_depth#3, beam(1), 10.00).
datum(beam_depth#4, beam(1), 10.12).
datum(beam_depth#5, beam(1), 9.94).
datum(beam_depth#6, beam(1), 9.75).
datum(beam_depth#7, beam(1), 10.08).
datum(beam_depth#8, beam(1), 9.90).
datum(beam_depth#9, beam(1), 9.00).
datum(beam_depth#10, beam(1), 8.25).
datum(beam_depth#11, beam(1), 8.00).
datum(beam_depth#12, beam(1), 8.06).
datum(beam_depth#13, beam(1), 7.93).
datum(beam_depth#14, beam(1), 8.14).
datum(beam_depth#15, beam(1), 8.00).

datum(web_thickness#1, beam(1), 0.615).
datum(web_thickness#2, beam(1), 0.415).
datum(web_thickness#3, beam(1), 0.340).
datum(web_thickness#4, beam(1), 0.350).
datum(web_thickness#5, beam(1), 0.318).
datum(web_thickness#6, beam(1), 0.292).
datum(web_thickness#7, beam(1), 0.252).
datum(web_thickness#8, beam(1), 0.240).
datum(web_thickness#9, beam(1), 0.575).
datum(web_thickness#10, beam(1), 0.365).
datum(web_thickness#11, beam(1), 0.288).
datum(web_thickness#12, beam(1), 0.285).
datum(web_thickness#13, beam(1), 0.245).
datum(web_thickness#14, beam(1), 0.248).
datum(web_thickness#15, beam(1), 0.230).
```

APPENDIX J

SAMPLE OUTPUT

A. COMPUTE WEIGHT

TEFA>>run(step_1, beam(1)).

cap_W=22.0 kip w=2.200000 kip/ft cap_L=10.0 ft

B. COMPUTE REACTIONS

TEFA>>run(step_2, beam(1)).

cap_R=11.0 kip cap_V=11.0 kip

C. COMPUTE MOMENT

TEFA>>run(step_3, beam(1)).

cap_M=27.500000 kip*ft

D. COMPUTE SECTION MODULUS

TEFA>>run(step_4, beam(1)).

cap_S=13.750000 in^3 cap_Fb=24.0 kip/in^2

E. SELECT BEAM

TEFA>>run(step_5, beam(1)).

w_sol=w8_17 d=8.0 in tw=0.230000 in

F. CHECK SHEAR

TEFA>>run(step_6, beam(1)).

shear=ok fv=5.978260 kip/in^2 cap_Fv=14.500000 kip/in^2

G. CHECK DEFLECTION

TEFA>>run(step_7, beam(1)).

defl=ok actual=0.310375 in allow=0.333333 in

APPENDIX K

DESCRIPTION OF COMPUTER PLATFORM

A. SYSTEM DEVELOPMENT INFORMATION

1. Computer platform: Macintosh IIsi.
2. Operating system: 6.07.
3. Processor: Motorola 68030, 20 mHz.
4. RAM: 5 megabytes.
5. Hard disk storage: 80 megabytes.

B. SYSTEM REQUIREMENTS

1. Computer platform: Mac II series.
2. Operating system: 6.05 or later.
3. RAM: 4 megabytes, minimum.
4. Hard disk: Highly desirable.

C. PROGRAMMING ENVIRONMENT

1. Model Management System: TEFA, version 1.10.
2. Programming language: Advanced A.I. Systems Prolog 2.0g. Prolog is the embedded language used by TEFA.

LIST OF REFERENCES

[1] Stoutemyer, David R., "Computing with Impure Numbers: Automatic Consistency Checking and Units Conversion Using Computer Algebra," *IEEE Transactions on Software Engineering*, Volume SE-3 No.6, pp. 478-480, November 1977.

[2] Bhargava, Hemant K., *Dimensional Analysis in Mathematical Modeling Systems, A Simple Numerical Method*, Naval Postgraduate School, Monterey, California, January 1991.

[3] Bhargava, Hemant K., *Building Models in TEFA: A Model Builder's Guide*, Naval Postgraduate School, Monterey, California, October 1990.

[4] Bradley, Gordon H., and Clemence, Robert D., "Model Integration with a Typed Executable Modeling Language," *Proceedings of the 21st Hawaii International Conference on System Sciences*, Kailu-Kona, Hawaii, 5-8 January 1988, Computer Society Press, Washington D. C.

[5] Markland, Robert E., and Sweigart, James R., *Quantitative Methods: Applications to Managerial Decision Making*, pp. 6-7, John Wiley & Sons, New York, 1987.

[6] Parker, Harry, *Simplified Design of Structural Steel*, John Wiley & Sons, New York, New York, 1974.

[7] Massey, B.S., *Measures in Science and Engineering: Their Expression, Relation, and Interpretation*, John Wiley and Sons, New York, New York, 1986.

[8] Langhaar, Henry, L., *Dimensional Analysis and Theory of Models*, John Wiley & Sons, New York, New York, 1951.

[9] Bhaskar, R., and Nigam, Anil, "Qualitative Physics Using Dimensional Analysis," pp. 73-111, *Artificial Intelligence*, Volume 45, Amsterdam, September 1990.

[10] Buckingham, E., "On Physically Similar Systems; Illustrations of the Use of Dimensional Equations," *Physical Review*, Vol. IV, No.4, pp. 345-376, 1914.

[11] Happ, William W., "Computer-Oriented Procedures for Dimensional Analysis," *Journal of Applied Physics*, Volume 38, Number 10, pp. 3918-3926, September 1976.

[12] Bradley, G., and Clemence, Robert D., Jr., "A Type Calculus for Executable Modelling Languages," *IMA Journal of Mathematics in Management*, pp. 277-291, 1987.

[13] Fourer, Robert, "Modeling Languages Versus Matrix Generator for Linear Programming," *Association for Computing Machinery: Transactions on Mathematical Software*, Vol. 9, No. 2, pp. 143-183, June 1983.

[14] Barnes, Christopher A., *Concept Hierarchies for Extensible Databases*, Master's Thesis, Naval Postgraduate School, Monterey, California, September 1990.

[15] U. S. Army Ballistic Research Laboratories REPORT NO. 1824, U.S Army Material Command, *A Computer Solution of the Buckingham Pi Theorem Using SYMBOLANG, a Symbolic Manipulation Language*, by Hirschberg, Morton, pp. 1-19, August 1975.

[16] Beyer, William H., *CRC Standard Mathematical Tables*, 28th edition, CRC Press, Inc., Boca Raton, Florida, 1987.

[17] Bhargava, Hemant K., *A Logic Model for Model Management: An embedded Languages Approach*, Ph.D. Thesis, University of Pennsylvania, Department of Decision Sciences, 1990.

[18] Dym, Clive L., and Ivey, Elizabeth S., *Principles of Mathematical Modeling*, Academic Press Inc., New York, New York, 1980.

[19] Philips Research Report 7, *Dimensional Analysis, Units and Rationalization*, by Vermeulen, R., pp. 432-441, 1952.

BIBLIOGRAPHY

Bhargava, Hemant K., and Kimbrough Steven O., "Model Management: An Embedded Languages Approach," *Proceedings of the Twenty-Third Annual Hawaii International Conference on Systems Sciences*, January 1990, revised, July 1990, forthcoming, Decision Support Systems, 1992.

Macagno, Enzo O., "Historico-critical Review of Dimensional Analysis," *Journal of the Franklin Institute*, Vol. 292, No.6, pp. 391-402, December 1971.

Pankhurst, R. C., "Alternative Formulation of the Pi-theorem," *Journal of the Franklin Institute*, Vol. 292, No.6, pp. 451-462, December 1971.

Stoutemyer, David R., "Dimensional Analysis, Using Computer Symbolic Mathematics," *Journal of Computational Physics*, Volume 24, pp. 141-149, 1977.

Van Driest, E. R., *Journal of Applied Mechanics*, Vol. 13, No.1, p. A34, March 1946.

INITIAL DISTRIBUTION LIST

1.	Defense Technical Information Center Cameron Station Alexandria, Virginia 22304-6145	2
2.	Library, Code 52 Naval Postgraduate School Monterey, California 93943-5002	2
3.	Curriculum Officer, Code 37 Computer Technology Naval Postgraduate School Monterey, California 93943-5000	1
4.	Hemant K. Bhargava Information Systems Department Code AS/Bh Naval Postgraduate School Monterey, California 93943-5000	5
5.	Gordon Bradley Operations Research Department Code OR/Bz Naval Postgraduate School Monterey, California 93943-5000	1
6.	Dan Dolk Information Systems Department Code AS/Dk Naval Postgraduate School Monterey, California 93943-5000	1
7.	LT Michael Elizondo, CEC, USNR ROICC Bldg. 332 Castle AFB Castle, California 95342-5000	2